

**Yee &  
Associates, P.C.**

4100 Alpha Road  
Suite 1100  
Dallas, Texas 75244

Main No. (972) 385-8777  
Facsimile (972) 385-7766

RECEIVED  
CENTRAL FAX CENTER

JUN 23 2005

## Facsimile Cover Sheet

To: Commissioner for Patents for Examiner Yuan, Almari Romero Group Art Unit 2176	Facsimile No.: 703/872-9306
From: Lourdes Perez Legal Assistant to Wing Yan Mok	No. of Pages Including Cover Sheet: 25
<p>Message:</p> <p>Enclosed herewith:</p> <ul style="list-style-type: none"><li>• Transmittal Document; and</li><li>• Appeal Brief.</li></ul> <p style="text-align: right;"><b>RECEIVED</b> OICE/IAP JUN 24 2005</p>	
Re: Docket No: AT9-98-920	Serial No. 09/306,189
Date: Thursday, June 23, 2005	
Please contact us at (972) 385-8777 if you do not receive all pages indicated above or experience any difficulty in receiving this facsimile.	<i>This Facsimile is intended only for the use of the addressee and, if the addressee is a client or their agent, contains privileged and confidential information. If you are not the intended recipient of this facsimile, you have received this facsimile inadvertently and in error. Any review, dissemination, distribution, or copying is strictly prohibited. If you received this facsimile in error, please notify us by telephone and return the facsimile to us immediately.</i>

**PLEASE CONFIRM RECEIPT OF THIS TRANSMISSION  
BY FAXING A CONFIRMATION TO 972-385-7766.**

## IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Cooper et al.

Serial No.: 09/306,189

Filed: May 6, 1999

For: Method and Apparatus for  
Converting Programs and Source  
Code Files Written in a Programming  
Language to Equivalent Markup  
Language Files

35525

PATENT TRADEMARK OFFICE  
CUSTOMER NUMBER§  
§  
§  
§  
§  
§

Group Art Unit: 2176

Examiner: Yuan, Almari Romero

Attorney Docket No.: AT9-98-920

Certificate of Transmission Under 37 C.F.R. § 1.8(a)

I hereby certify this correspondence is being transmitted via facsimile to  
the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-  
1450, facsimile number (703) 872-9306 on June 23, 2005.

By:

Lourdes Perez

TRANSMITTAL DOCUMENT

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

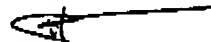
Sir:

ENCLOSED HEREWITH:

- Appeal Brief (37 C.F.R. 41.37).

A fee of \$500.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

Respectfully submitted,



Wing Yan Mok  
Registration No. 56,237  
AGENT FOR APPLICANTS

Duke W. Yee  
Registration No. 34,285  
ATTORNEY FOR APPLICANTS

YEE & ASSOCIATES, P.C.  
P.O. Box 802333  
Dallas, Texas 75380  
(972) 385-8777

Docket No. AT9-98-920

RECEIVED  
CENTRAL FAX CENTER

PATENT

JUN 23 2005

## IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Cooper et al.

Serial No. 09/306,189

Filed: May 6, 1999

For: Method and Apparatus for  
Converting Programs and Source  
Code Files Written in a Programming  
Language to Equivalent Markup  
Language Files§  
§  
§  
§  
§  
§  
§

Group Art Unit: 2176

Examiner: Yuan, Almari Romero

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450**Certificate of Transmission Under 37 C.F.R. § 1.8(a)**I hereby certify this correspondence is being transmitted via  
facsimile to the Commissioner for Patents, P.O. Box 1450,  
Alexandria, VA 22313-1450, facsimile number (703) 872-9306  
on June 23, 2005.

By:

  
Lourdes Perez

06/24/2005 HLE333 00000035 090447 09306189

01 FC:1402 500.00 DA

**APPEAL BRIEF (37 C.F.R. 41.37)**

This brief is in furtherance of the Notice of Appeal, filed in this case on April 27, 2005.

The fees required under § 41.20(B)(2), and any required petition for extension of time for filing this  
brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.Appeal Brief Page 1 of 23  
Cooper et al. - 09/306,189

**REAL PARTY IN INTEREST**

The real party in interest in this appeal is the following party: International Business Machines Corporation.

**RELATED APPEALS AND INTERFERENCES**

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

**STATUS OF CLAIMS**

**A. TOTAL NUMBER OF CLAIMS IN APPLICATION**

Claims in the application are: 6-11, 17-22, and 25.

**B. STATUS OF ALL THE CLAIMS IN APPLICATION**

1. Claims canceled: 1-5, 12-16, 23-24, and 26.
2. Claims withdrawn from consideration but not canceled: None.
3. Claims pending: 6-11, 17-22, and 25.
4. Claims allowed: None.
5. Claims rejected: 6-11, 17-22, and 25.
6. Claims objected to: None.

**C. CLAIMS ON APPEAL**

The claims on appeal are: 6-11, 17-22, and 25.

**STATUS OF AMENDMENTS**

There are no amendments after final rejection.

**SUMMARY OF CLAIMED SUBJECT MATTER****A. CLAIMS 6, 17, and 25 - INDEPENDENT**

Independent claims 6, 17, and 25 of the present invention are directed to a method of dynamically translating an application program into a markup language file, the method comprising the computer-implemented steps of: executing said application program; parsing a document type definition file for a markup language; during execution of said application program, selecting an element defined in the document type definition file based on a routine called by said application program; and writing the selected element to a markup language file to form a translation (Specification, page 23 to page 25).

**B. CLAIMS 7 and 18 - DEPENDENT**

Dependent claims 7 and 18 of the present invention are directed to a method of dynamically translating an application program into markup language file, in which the element comprises an attribute list corresponding to parameters for the routine (Figure 12 element 1208).

**C. CLAIMS 8 and 19 - DEPENDENT**

Dependent claims 8 and 19 of the present invention are directed to a method of dynamically translating an application program into markup language file, in which the selected element written to the markup language file comprises an attribute list corresponding to values for the parameters passed to the routine (Specification, page 20).

**D. CLAIM 17 - INDEPENDENT**

Independent claim 17 of the present invention is directed to a data processing system for dynamically translating an application program into a markup language file that comprises executing means for executing an application program (Figure 2, element 202); parsing means for parsing a document type definition file for a markup language (Figure 4B, element 404, Figure 6, element 604); selecting means for selecting an element defined in the document type definition file based on a routine called by the application program (Figure 4B, element 404, Figure 6, element 604); and writing means for writing the selected element to a markup language file to form a translation (Figure 4B, element 404, Figure 6, element 612).

**GROUND OF REJECTION TO BE REVIEWED ON APPEAL**

The grounds of rejection to be reviewed on appeal are:

Claims 6-11, 17-22, and 25 are rejected as being allegedly unpatentable over Meltzer et al. (U.S. Patent No. 6,226,675 B1) in view of Villacis et al., "A Web Interface to Parallel Program Source Code Archetypes," 1995, ACM, Inc., pages 1-16 under 35 U.S.C. §103(a).

## ARGUMENT

### **I. 35 U.S.C. § 103(a), Alleged Obviousness, Claims 6-11, 17-22, and 25**

The Final Office Action rejects claims 6-11, 17-22, and 25 under 35 U.S.C. § 103(a) as being allegedly unpatentable over Meltzer et al. (U.S. Patent No. 6,226,675 B1) in view of Villacis et al., "A Web Interface to Parallel Program Source Code Archetypes", 1995, ACM, Inc., pages 1-16. This rejection is respectfully traversed.

As to claim 6, 17, and 25, the Final Office Action states in the Response to Arguments section:

Applicant argues that Villacis does not disclose "~~during execution of said application program, selecting an element defined in the document type definition file based on a routine called by said application program~~".

Villacis discloses the conversion of source codes or programs into WWW hypertext documents. A special compiler is used to examine the source code and discover all subroutine call sites (routines) to automatically build the hypertext links to the appropriate subroutine definitions (see Abstract and see pages 4, 7-8, and 14).

Villacis on page 8, 1<sup>st</sup> paragraph teaches ~~the user can publish a program source as an interactive document by generating a hypertext version of a source code; wherein the source code can be a program of Fortran dialect~~. Each program subroutine can correspond to a hypertext link. Furthermore, Villacis on page 11, 3<sup>rd</sup> paragraph teaches a compiler and runtime system to ~~run a series of conversion programs to produce HTML documents~~. Villacis on page 12, Figure 8 shows ~~conversion process from a source code into HTML~~ by the use of a compiler and a Sage++ to run and parse the source code.

Meltzer does disclose the "selecting an element defined in the document type definition file" as described above in rejected claim 6. (Emphasis added)

Final Office Action dated January 27, 2005, page 4-5.

Independent claim 6, which is representative of claims 17 and 25 with regard to similarly recited subject matter, recites:

6. A method of dynamically translating an application program into a markup language file, the method comprising the computer-implemented steps of:
  - executing said application program;
  - parsing a document type definition file for a markup language;
  - ~~during execution of said application program, selecting an element defined in the document type definition file based on a routine called by said application program;~~ and
  - writing the selected element to a markup language file to form a

translation. (Emphasis added)

Neither Meltzer nor Villacis teaches or suggests the features emphasized above. As discussed in the abstract, Meltzer teaches a method of exchanging self-defining electronic documents, such as XML based documents, between trading partners without custom integration. The definitions of the electronic business documents, called business interface definitions, are posted on the Internet, or communicated to members of the network. The business interface definition tells potential trading partners the services the company offers and the documents to use when communicating with such services. Participants are programmed by the composition of the input and output documents, coupled with interpretation information in a common business library, to handle the transaction in a way, which closely parallels the way in which paper businesses operate.

Meltzer does not teach that during execution of an application program, selecting an element defined in a document type definition file based on a routine called by the application program, as recited in claims 6, 17, and 25 of the present invention. At column 25, line 52 to column 26, line 9, Meltzer teaches a Java to XML event generator that receives a stream of events and translates selected ones to present a Java object as an XML document. The "selected ones" as referred to in Meltzer, are selected Java events generated by the Java walker as the Java walker walks the DOM object. At column 24, lines 31-45, Meltzer teaches an element event generator whose listeners are only interested in events for particular types of elements. For example, in an HTML document, the listener may only be interested in ordered lists, that is only the part of the document between <OL> and </OL> tags. Thus, Meltzer's selected Java events are merely events that are selected for particular types of elements within a document. The selected Java events are not elements selected based on a routine called by the application program. Thus, Meltzer selects an event based on whether the element is a particular type of element within a document, rather than based on whether the event is a routine called by an application program.

In addition, as Meltzer later teaches, each object of the selected Java events that the Java walker walks becomes an element. Within each element, each embedded method becomes an element whose content is the value returned by invoking the method. Thus, Meltzer explicitly teaches that each embedded method of an object becomes an element, whether the method is called by the application program or not. Therefore, Meltzer selects an element based on the criteria that a

particular routine is embedded within an object, as opposed to a routine called by the application program during the execution of the application program as recited in claims 6, 17, and 25 of the present invention.

It is clear that with Meltzer, every method in the object will be selected as an element. This is contrary to the presently claimed invention, which recites that an element is selected based on a routine called by the application program. Only an element corresponding to a routine called by the application program is selected. Thus, the present invention has an advantage over Meltzer in that the present invention includes an ability that Meltzer fails to address: the ability to select elements based on a routine called by the application program. Meltzer, on the other hand, only teaches selecting events for particular types of elements and selecting each method of an object of the selected Java events. Meltzer does not teach or suggest, in any section of the reference, the ability to select an element based on a routine called by the application, as recited in claims 6, 17, and 25 of the present invention, because Meltzer is only interested in translating particular elements of a document to a format of a host (column 23, lines 45-46). There is no need for Meltzer to select an element based on a routine that is called by an application program, since Meltzer is not interested in the routine called by that application program. Therefore, a person of ordinary skill in the art would not be led to modify Meltzer's listener to reach the presently claimed invention simply because Meltzer does not teach or suggest selecting an element based on a routine called by an application.

Meltzer actually teaches away from the presently claimed invention because Meltzer explicitly teaches selecting events for particular types of element within a document and that each embedded method within an object becomes an element, as opposed to selecting an element based on a routine called by the application. Absent the Examiner pointing out some teaching or incentive to implement Meltzer in this manner, one of ordinary skill in the art would not be led to modify Meltzer to reach the presently claimed invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify Meltzer in this way, the presently claimed invention can be reached only through an improper use of hindsight using the Applicants' disclosure as a template to make the necessary changes to reach the presently claimed invention. Thus, Meltzer does not teach or suggest during execution of an application program, selecting an element defined in a document type definition file based on a routine called by the application program, as recited in claims 6, 17, and 25

In addition, the Examiner admits in the Final Office Action that Meltzer fails to teach the feature of during execution of an application program, selecting an element defined in the document type definition file based on a routine called by said application program. However, the Examiner alleges that Villacis teaches these features. As discussed in the Abstract, Villacis is directed to a set of tools for annotating and exploring program source code on the World Wide Web. The tools in Villacis are part of a project to build an electronic textbook for parallel programming that exploits Caltech Archetypes model of program construction. The tools provide a way for Fortran, HPF and C++ programmers to add special annotations to source code that allow the source code to be converted automatically into WWW hypertext documents. In addition, special compiler based tools examine the source code and discover all subroutine call sites and automatically build the hypertext links to the appropriate subroutine definitions. Thus, Villacis is a system for generating textbook templates, in a hypertext document format, for source code.

The Final Office Action alleges that Villacis teaches the feature of during execution of an application program, selecting an element defined in the document type definition file based on a routine called by said application program on page 8, 1<sup>st</sup> paragraph. In this section, Villacis merely teaches a mechanism, named Mccp, which accepts programs written in Fortran or C++ flavor dialects and generates a hypertext version of the source code. The result is a forms-based HTML document tree. Thus, Villacis teaches a mechanism that explores the source code of a program and publishes the program source as an interactive document, such that the user may selectively view definitions and structures of the program. Nowhere in this section, or in any other section, of the reference does Villacis teach or suggest that the generation of the hypertext document occurs during execution of an application program.

To the contrary, Villacis's mechanism operates on the source code of the application program when the source code is compiled. On page 8 of the reference, Villacis teaches that a complete subroutine call graph is used to generate the hyperlinks between each invocation site and the appropriate routine definitions. On page 7 of the reference, Villacis teaches that the complete static call graph and class hierarchy data base is built by extracting data from an intermediate pass of the compiler. Thus, Villacis's mechanism uses the complete static call graph that was built during compilation of the source code to generate the hyperlinks. Villacis's mechanism does not generate the hyperlinks during execution of the source code.

In addition, nowhere in the reference does Villacis teach or suggest selecting an element defined in the document type definition file based on a routine called by said application program. Villacis only teaches on page 8 that hypertext links are generated between each invocation site and appropriate routine definitions based on the complete subroutine call graph built by the compiler, not a routine called by an application program. Therefore, Villacis does not teach or suggest the features of claims 6, 17, and 25 of the present invention.

The Final Office Action further alleges that Villacis teaches the features of during execution of an application program, selecting an element defined in the document type definition file based on a routine called by said application program on page 11, 3<sup>rd</sup> paragraph and on page 12, Figure 8. Figure 8 of the reference is shown below:

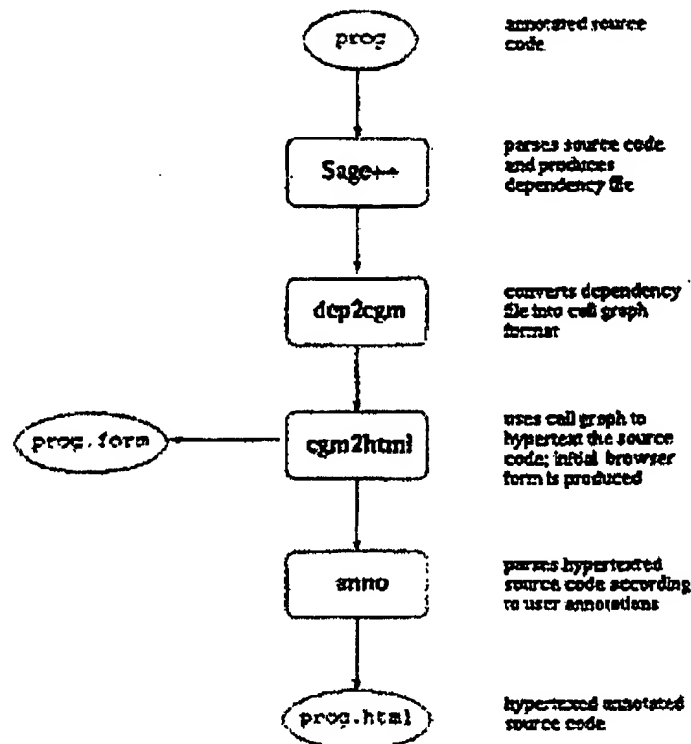


Figure 8: Conversion from Annotated Source Code to Hypertext

As shown in Figure 8, Villacis teaches a compiler system and a runtime system. In the compiler system, during the compiling phase, the author of the source code first annotates his code and runs his source code through Sage++, which parses the source code and produces a dependency file that contains a parse tree of the source code. The parse tree is then converted by dep2cgm to a call graph, which is used by cgm2html to hypertext the source code. The hypertexted source code is passed through the Anno program where it is parsed according to the author's annotations and results in an initial base HTML document, prog.html. However, none of the conversion steps occur during execution of an application program. Rather, the conversion steps occur during the compiling phase in the compiler. In addition, Villacis does not teach or suggest that the hypertext document is generated based on a routine called by an application program. Instead, the hypertext document is generated based on a call graph. Therefore, Villacis does not teach or suggest the features of claims 6, 17 and 25 of the present invention.

On the other hand, in the runtime system, Villacis teaches a Mccp browser program that permits the user to browse the HTML documents produced from the source code during the compiler phase. During the runtime phase, Villacis discusses the creation of new documents when a user clicks on a form button or expand link in the initial document. These operations are performed on the HTML document version of the source code. They do not select an element defined in a document type definition file based on a routine called by an application program during execution of the application program. To the contrary, nowhere in Villacis is either the source code or the HTML document version of the source code ever executed. This is because Villacis is only concerned with generating templates of the source code for Villacis' "textbook" so that they may be browsed using the Mccp browser. Villacis is not concerned with executing code and furthermore, is not concerned with selecting an element from a document type definition file based on a routine called by an application program during execution of the application program. Thus, while Villacis may teach a runtime environment, this runtime environment is only taught as providing a browser by which to view the HTML document versions of the source code and has nothing to do with the selection of elements defined in a document type definition file based on a routine called by an application during execution of the application, as recited in claims 6, 17, and 25 of the present invention.

Furthermore, the Final Office Action alleges that it would have been obvious to a person of ordinary skill in the art at the time the invention was made to have modified Villacis into Meltzer to provide a program containing subroutine call sits and convert this program into a hypertext document as taught by Villacis and incorporated into the converting of programming structures into an XML document of Meltzer, in order to help programmers turn source code into hypertext documents in scalable parallel computer environment. Appellants respectfully disagree.

Meltzer is directed to a Java walker that walks a tree of Java bean components and translates selected Java events to present a Java object as an XML document (column 25, line 52 to column 26, line 9). Villacis is directed to converting a program source code during compilation to a hypertext document for users to view the definitions and structures of the source code. Neither Meltzer nor Villacis teaches or suggests that an element is selected during execution of an application program based on a routine executed by the program. The Java walker of Meltzer walks over all publicly accessible fields and methods of an object, not a routine that is executed by an application program. The Mccp of Villacis translates source code into a hypertext document during compilation of the source code, not during execution of the source code. Therefore, a person of ordinary skill in the art would not have been led to modify the Mccp of Villacis to incorporate the Java walker of Meltzer, because the resulting combination would not be selecting an element defined in a document type definition file based on a routine called by an application program during execution of the application program. Rather, the resulting combination would be translating all publicly accessible fields and methods of a source code to a hypertext document during compilation of the source code. Therefore, a person of ordinary skill would not have been led to make the combination as alleged by the Examiner.

In view of the above, Appellants respectfully submit that neither Meltzer nor Villacis teaches or suggests the features of claims 6, 17, and 25. At least by virtue of their dependency on claims 6 and 17 respectively, neither Meltzer nor Villacis teaches or suggests the features of dependent claims 7-11 and 18-22. Accordingly, Appellants respectfully request the withdrawal of the rejection of claims 6-11, 17-22, and 25 under 35 U.S.C. § 103(a).

In addition, neither Meltzer nor Villacis teaches or suggests any of the specific features set forth in the dependent claims 7-11 and 18-22. With regard to dependent claim 7, which is representative of claim 18 with regard to similarly recited subject matter, neither Meltzer nor

Villacis teaches or suggests that an element comprises an attribute list corresponding to parameters for a routine that is executed. The Final Office Action alleges that Meltzer teaches these features at column 76, lines 33-67, which reads as follows:

A compiler takes the purchase order definition and generates several different target forms. All of these target forms can be derived through "tree to tree" transformations of the original specification. The most important for this example are:

- (a) the XML DTD for the purchase order
- (b) a JAVA Bean that encapsulates the data structures for a purchase order (the JAVA classes, arguments, datatypes, methods, exception structures are created that correspond to information in the Schema definition of the purchase order).
- (c) A "marshaling" program that converts purchase orders that conform to the Purchase Order DTD into a Purchase Order JAVA Bean or loads them into a database, or creates HTML (or an XSL style sheet) for displaying purchase orders in a browser.
- (d) An "unmarshaling" program that extracts the data values from Purchase Order JAVA Beans and converts them into an XML document that conforms to the Purchase Order DTD.

Now back to the scenario. A purchasing application generates a Purchase Order that conforms to the DTD specified as the service interface for a supplier who accepts purchase orders.

The parser uses the purchase order DTD to decompose the purchase order instance into a

stream of information about the elements and attribute values it contains. These "property sets" are then transformed into corresponding JAVA event objects by wrapping them with JAVA code. This transformation in effect treats the pieces of marked-up XML document as instructions in a customer programming language whose grammar is defined by the DTD. These JAVA events can now be processed by the marshaling applications generated by the compiler to "load" JAVA bean data structures.

(Column 76, lines 33-67, Meltzer)

In the above section, Meltzer teaches an unmarshaling program that extracts data values from a purchase order Java bean and converts them into an XML document that conforms to the Purchase Order. However, Meltzer does not teach an element that comprises an attribute list corresponding to parameters for the routine. While Meltzer teaches a Java bean that encapsulates data structures of a purchase order that include arguments, Meltzer does not teach an element that comprises an attribute list of these arguments (parameters for the routine). To the contrary, Meltzer teaches extracting only data values from the Java bean to convert to an XML document. When converting from XML to Java using Meltzer's marshaling program, the parser parses the purchase

order XML document according to the document type definition to retrieve elements and attributes of the XML document. However, when converting from Java to XML, Meltzer's unmarshaling program only calls for extracting data values from the Java bean, not parameters for the routine, as recited in claims 7 and 18 of the present invention.

In addition, at column 25, line 52 to column 26, line 9, Meltzer teaches that each embedded method becomes an element whose content is the value returned by invoking the method. Meltzer does not teach an element comprising an attribute list of parameters for the routine, since the parameters for the routine are inputs to the routine. Rather, Meltzer teaches an element comprising a value returned by invoking the method. The value returned is output of the routine. In Java, as known by a person of ordinary skill in the art, "parameters" for a routine carries a meaning of input parameters or input arguments of a routine. Thus, "parameters" for a routine are commonly associated with inputs to a routine. To the contrary, a "returned value" is commonly known as an output value from invoking a routine. In the case of Java, only one returned value is normally associated for a routine, which represents an output of the routine. Therefore, Meltzer does not teach an element comprising an attribute list of "parameters" for the routine, since Meltzer only teaches an element whose content is output of a routine, not parameters (inputs) for the routine.

Villacis also does not teach an element that comprises an attribute list corresponding to parameters for a routine that is executed. Villacis merely teaches translating a program source code definitions and structures into a hypertext document during compilation of the program. Since Villacis does not teach or suggest executing the source code, Villacis does not and would not teach an element that comprises an attribute list corresponding to parameters for a routine that is executed. In addition, Villacis is only concerned with presenting the definitions and structures of the source code to the users, rather than parameters for a routine that is executed. Therefore, Villacis also does not teach or suggest the features of claims 7 and 18 of the present invention.

With regard to dependent claim 8, which is representative of claim 19 with regard to similarly recited subject matter, Meltzer does not teach that a selected element that is written to a markup language file comprises an attribute list corresponding to values for parameters passed to the routine. The Final Office Action alleges that Meltzer teaches these features at column 76, lines 33-67, which is reproduced above. However, as described above, Meltzer teaches that only data values are written to the XML document by the unmarshaling program. The data values of Meltzer are values returned by invoking the routine, not values for the parameters passed to the routine.

Meltzer is only interested in placing returned data as a result of invoking a routine into the XML document, Meltzer is not interested in placing values passed to the parameters for the routine in the XML document, as recited in claim 8 of the presently claimed invention. The data values returned by invoking a routine are outputs of the routine. To the contrary, values passed to the routine for the parameters are inputs to the routine. Therefore, one of ordinary skill in the art would not be led to modify Meltzer's teaching to reach the presently claimed invention, because Meltzer does not teach the selected element written to the markup language file comprises an attribute list corresponding to values for the parameters passed to the routine as recited in claims 8 and 19 of the present invention.

Villacis also does not teach a selected element that is written to a markup language file comprises an attribute list corresponding to values for parameters passed to the routine. As discussed above in arguments presented in claims 7 and 18, Villacis merely teaches translating a program source code definitions and structures into a hypertext document during compilation of the program source code. Since Villacis does not execute the source code during compilation, no values for parameters are passed to the routine. In addition, Villacis is not concerned with presenting values for parameters passed to the routine to users. Villacis is only concerned with presenting definitions and structures of the source code to the users. Therefore, Villacis also does not teach or suggest the features of claims 8 and 19 of the present invention.

Thus, in addition to their dependency on independent claims 6 and 15, neither Meltzer nor Villacis teaches or suggests the features of dependent claims 7-11 and 18-22 of the present invention. Accordingly, Appellants respectfully request the withdrawal of the rejection of claims 7-11 and 18-22 under 35 U.S.C. §103(a).

**CONCLUSION**

In view of the comments above, Appellants respectfully submit that the rejections of claims 6-11, 17-22, and 25 are overcome. Accordingly, it is respectfully urged that the rejection of claims 6-11, 17-22, and 25 not be sustained.



---

Wing Y Mok  
Reg. No. 56,237  
YEE & ASSOCIATES, P.C.  
PO Box 802333  
Dallas, TX 75380  
(972) 385-8777

**CLAIMS APPENDIX**

The text of the claims involved in the appeal are:

6. A method of dynamically translating an application program into a markup language file, the method comprising the computer-implemented steps of:
  - executing said application program;
  - parsing a document type definition file for a markup language;
  - during execution of said application program, selecting an element defined in the document type definition file based on a routine called by said application program; and
  - writing the selected element to a markup language file to form a translation.
7. The method of claim 6 wherein the element comprises an attribute list corresponding to parameters for the routine.
8. The method of claim 6 wherein the selected element written to the markup language file comprises an attribute list corresponding to values for the parameters passed to the routine.
9. The method of claim 6 wherein the application program is written in Java programming language.
10. The method of claim 9 wherein the routine is an extended class method.
11. The method of claim 9 wherein the routine is a Graphics class method.

17. A data processing system for dynamically translating an application program into a markup language file, the data processing system comprising:
- executing means for executing an application program;
  - parsing means for parsing a document type definition file for a markup language;
  - selecting means for selecting an element defined in the document type definition file based on a routine called by the application program; and
  - writing means for writing the selected element to a markup language file to form a translation.
18. The data processing system of claim 17 wherein the element comprises an attribute list of parameters for the routine.
19. The data processing system of claim 17 wherein the selected element written to the markup language file comprises an attribute list of values for the parameters passed to the routine.
20. The data processing system of claim 17 wherein the application program is written in Java programming language.
21. The data processing system of claim 20 wherein the routine is an extended class method.
22. The data processing system of claim 20 wherein the routine is a Graphics class method.

25. A computer program product on a computer readable medium for use in a data processing system for dynamically translating an application program into a markup language file, the computer program product comprising:

first instructions for executing an application program;

second instructions for parsing a document type definition file for a markup language;

third instructions for selecting an element defined in the document type definition file based on a routine called by the application program; and

fourth instructions for writing the selected element to a markup language file to form a translation.

**EVIDENCE APPENDIX**

There is no evidence to be presented.

**RELATED PROCEEDINGS APPENDIX**

There are no related proceedings.